The art of dialog creation for FOnline: 2238

Written by Ghosthack

Revision 1

2010-07-05

Introduction

In this guide I will try to touch on many of the aspects involved in a creating a dialog for FOnline: 2238. We will begin with creating a simple dialog tree without much finesse and then from there try to in a step-by-step manner create a fully rich NPC dialog capable of being part of a quest.

This guide assumes you've never worked with the FOnline engine, therefore a few concepts that people who are used to working with the engine already know might be explained in overly detailed, you are advised to skip those sections.

We won't ponder much about how to write good dialogs language wise, as this is not within the scope of this tutorial. This is mostly a technical tutorial.

Prerequisites

The most important thing you'll need when creating a dialog is of course the DialogEditor, although in theory it's possible to create a dialog in a text editor, doing so doesn't make a whole lot of sense as the dialog format is not very clear reading unless you're a Vulcan or perhaps an artificial intelligence.

Once you have the DialogEditor and the few data files needed for it to run, you'll just have to launch it to and wait for it to fully load.

Worth noting is that although the program might run on a Linux distribution or on Mac OSX via WINE (WINE is a compatibility layer for running Windows program on other operating systems, see http://winehq.org*) it might not work quite as well as it does on Windows.*

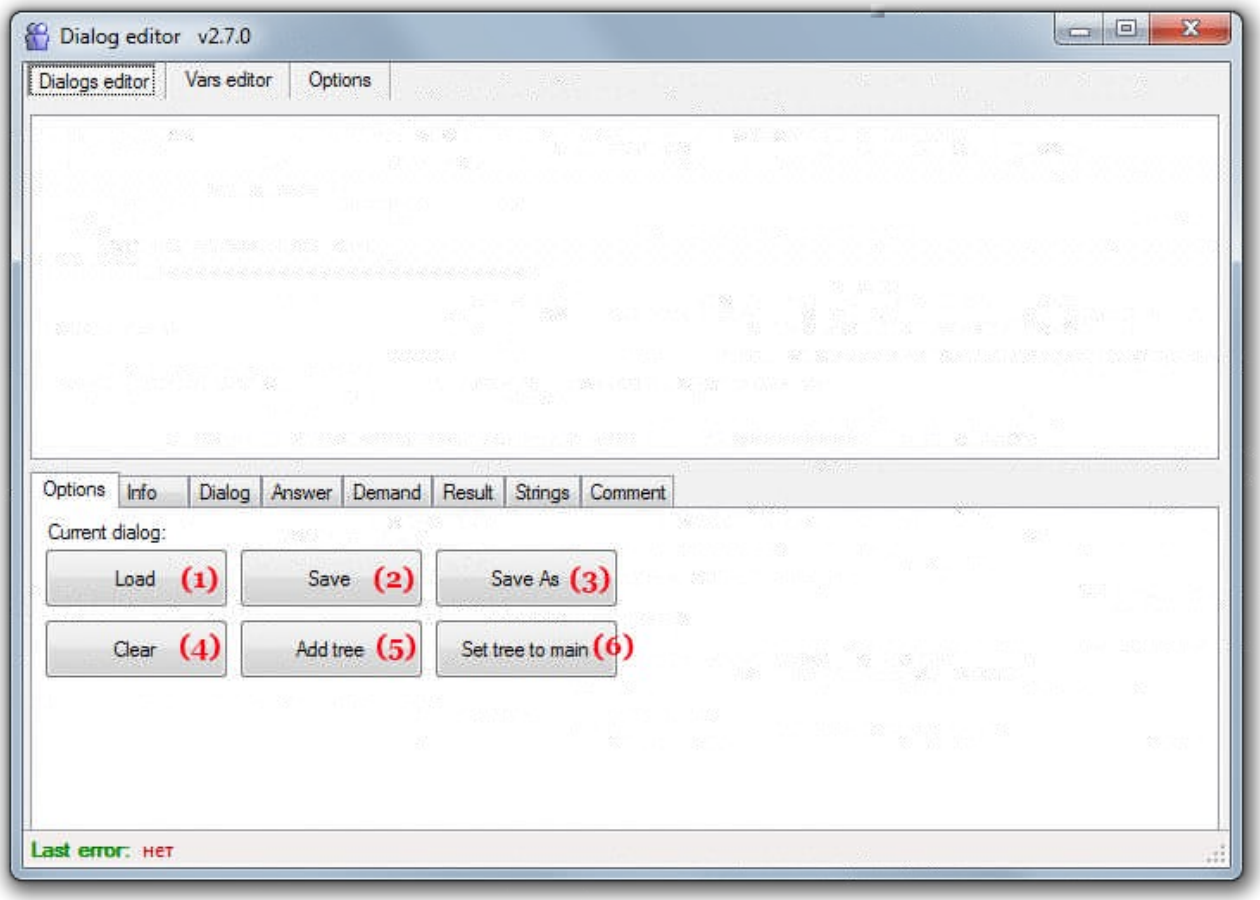
Legacy

The FOnline engine didn't come about out of thin air so it's not hard to imagine that some of the older file formats that were used in the original games were inherited if not verbatim, at least in spirit as the same kinds of problems are tackled, although the engine is more advanced and far more adapted to modern development practices.

The message string format “{100}{}{Text String}” is for example the same as in the original Fallouts and so are much of the terminology. A critter for example, refer to both an NPC and a PC. If you're uncertain of some term that is not mentioned anywhere, looking around on http://falloutmods.wikia.com can be of great help.

The DialogEditor

Opening the editor you will be prompted with an empty white area and a couple of tabs. The white area is where the actual dialog tree is shown. We will now proceed to see what is what.



(1) **Load –** Load a dialog from file.

(2) **Save** – Save current dialog to file.

(3) **Save as** – Save current dialog with new name.

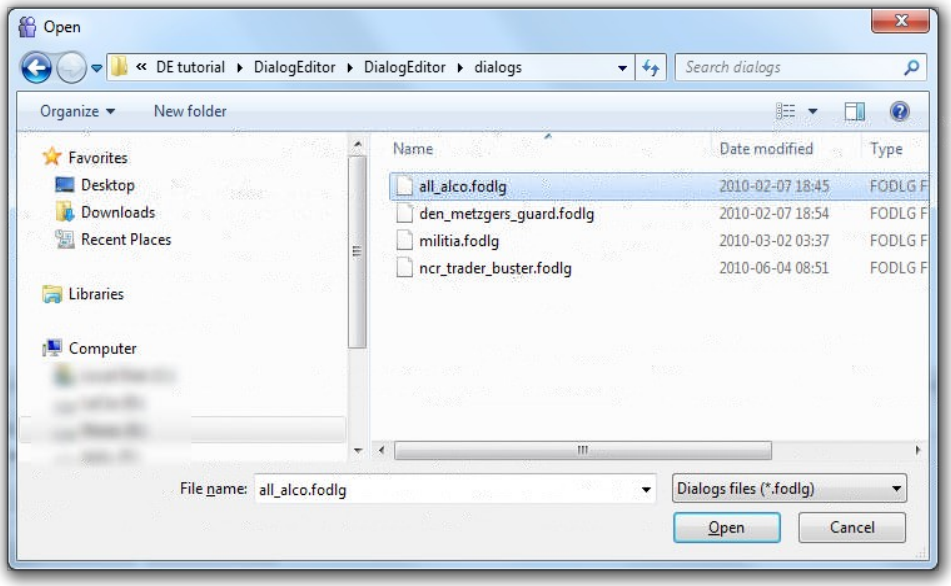
(4) **Clear** – Delete dialog and start from scratch.

(5) **Add tree –** Add a dialog tree in a new language, the target language is denoted with exactly **four characters**.

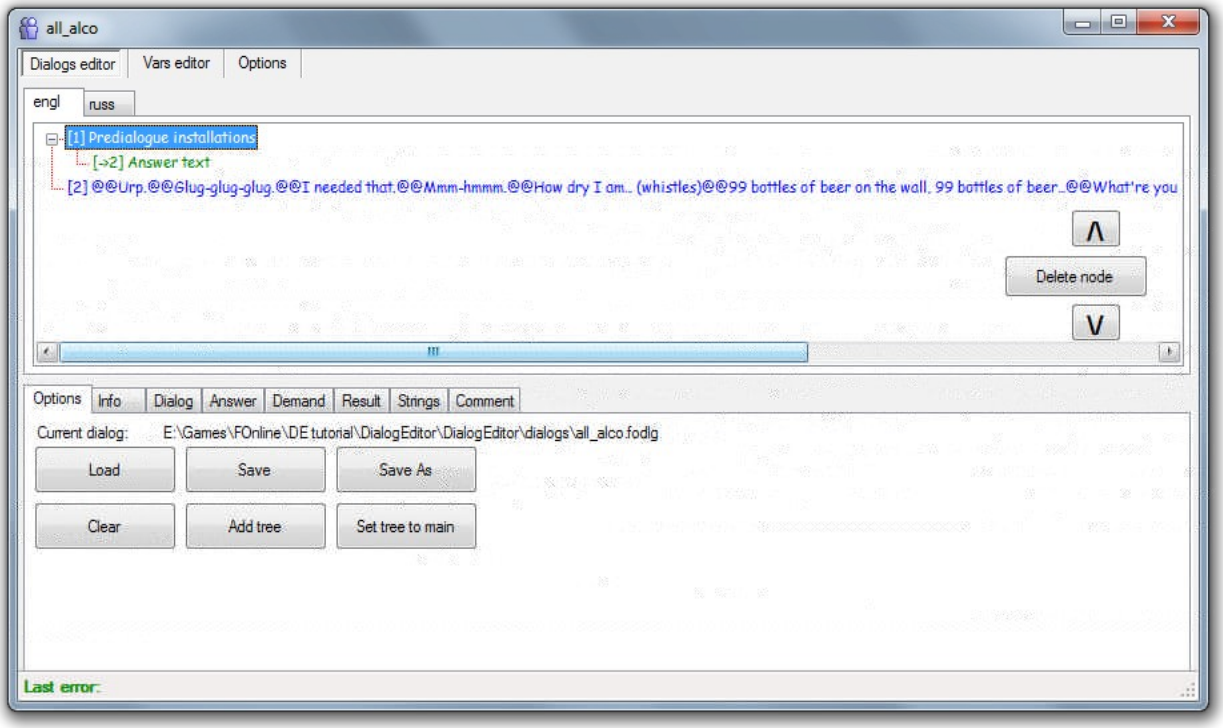
(6) **Set tree to main** – Sets currently selected tree (language) to the primary one.

Creating your first dialog

Although it is possible to create a dialog entirely from scratch, it's better to open an existing dialog to use as template, so, now we will open the dialog named all_alco.fodlg which should reside in the dialogs folder, which is located in the same folder as the DialogEditor.



After loading the dialog file, the tree will be displayed.



After this there are a couple of things worth noting, what we are now seeing is the English tree (the only one used) with two *dialog nodes*. The node selected in the image above is the so called *Pre-dialog node* this node is the first one executed by the engine. When a dialog is initiated in the game, there has to be a way for the dialog designer to decide the initial state of the dialog tree, this is where it happens.

The underlying sub-node is what is called *an answer node* this is what the player in the game sees as his answer options. In the pre-dialog node this acts in a peculiar way though, because it actually decided which node is going to be the initial one.

The tree is built-up of a few fundamental building blocks, the node (which is the NPC text displayed in the dialog), the answer, the demand and the result. I've explained the first

two, but right now we won't concern ourselves with the last ones until a bit later.

The number inside of the brackets, is the node dialog id used for internal reference. This is used when jumping between nodes from either a pre-dialog node or a normal answer node.

Editing the dialog tree

Now let's get down to business. We want to create our own unique dialog, and the first step in doing so, would be to remove the current floater node, the only dialog node except the pre-dialog node (which should always be present). This can be done by selecting the node and clicking the button *Delete node*. Now we only got a pre-dialog node pointing to nothing left.

For the sake of this tutorial, I've decided to make a certain “Mysterious Stranger” NPC, however this is not at all needed, you may enter whatever text or properties that you want straight away, I'm just trying to create some guidance.

It's pretty pointless to have a dialog which contains nothing, so let's click the *Dialog tab* and enter some text in the textbox. Let's say we've decided that this is going to be a floater that only pops up when the charisma of the player is under four. As we are not trying to create a novel or be good writers in this tutorial, just enter “You're an ugly one. Get away from me” or something similar. The dialog number should be correct, make sure that Auto Id is checked, as the editor will then assign ids automatically, something you almost always want it to. Then click *Add dialog*.

You can also edit nodes by right-clicking them in the dialog tree.

No need to swim if you can float

Creating a floater is very simple, you just make sure there is no answer nodes on the dialog node that you want as a floater. The game will automatically detect this and make it a floater.

Demanding results

If it weren't possible to create some kind of conditional jumps or conditions for selecting various answer options, the possibility of making rich dialogs wouldn't exist. Almost every dialog in FOnline has at least some kind of condition, and this is exactly what demands are for.

We want the insulting floater to be said only when charisma of the player is under four, to achieve this, we click the *demand tab* and make sure the answer node on the pre-dialog node is selected (“[2] Answer text”).

The screen presented might seem a bit confusing at first, but it's pretty straightforward when you've used it a couple of times. First you need what type of value is demanded, in this case we demand a *parameter*, after clicking the radio button, we click the *parameter dropdown menu* and select **ST_CHARISMA**, then we proceed to click the operand button until it shows “<” (less than), then we proceed to type in 4 in the *Demand value* field. Make sure that *Player* is selected, or else it will check against the NPC's charisma. Press *Add*.

Now we of course only have one additional dialog node and one possible entry jump from the pre-dialog node. Let's add an additional node. Press the *Dialog tab* and create a new node with the text “Hello there stranger, what do you want?”. We want this node to be the starting **unless** the player meets the condition of having a charisma of less than four. Therefore we add another answer to the pre-dialog node, this node **has to be after** the first one with the demand. When creating the answer, we select the new dialog node's id as the *On dialog* number.

Worth noting is that you can select some other options here, for example close the dialog on answer, open a barter dialog or initiate an attack on the player (because of insulting remarks for example). It's recommended to use these sparingly and it's good design to make it absolutely clear that said lines will initiate an attack or terminate the conversation.

Now let's create some additional nodes which will be the NPCs replies to our answer nodes which we will create soon. The work flow requires that you create the dialog (NPC's) nodes before creating the player answers.

To showcase how results work, let's create some answer nodes to our new dialog node, but before doing this we of course have to create some dialog nodes to redirect those answers to. Let's create them. “Hmm, let's see... I'll give you 50 caps.” (bear with me) and “Get out of my face then”. After that we create three answers, make sure that you've clicked on the “Hello there stranger, what do you want?” node, otherwise the answer nodes won't be created under this node.

We want to connect the answer “ I've found a combat knife, how much is it worth to ya?” to the caps giving node. Of course this answer should only be available if the player actual has a combat knife, so we add a demand node for the *item* **PID_COMBAT_KNIFE** and operand “>” (greater than) with the demand value 0.

Now we also want to add some caps to the players inventory for his efforts. This is where result comes in. Let's click on the *result tab*. This screen looks a lot like the demand one, but has different operands, that *does something* to the value you've selected.

These operands work like this: The value will be = value<operand><result value> an example would be If you had 50 bottle caps in your inventory already, and had selected the operand *, and then proceed to enter 2 as result value, you will then have 50*2=100 caps.

We create a new answer node to the “Hmm, let's see... I'll give you 50 caps.” node, “That will do, thanks, bye!”. In this case we want to add a number of caps to the inventory of the player, so we select the item **PID_BOTTLE_CAPS**, the “+” operand and enter 50. Make sure that *Result for* points to Player. Then we'll add another result that removes a **PID_COMBAT_KNIFE**.

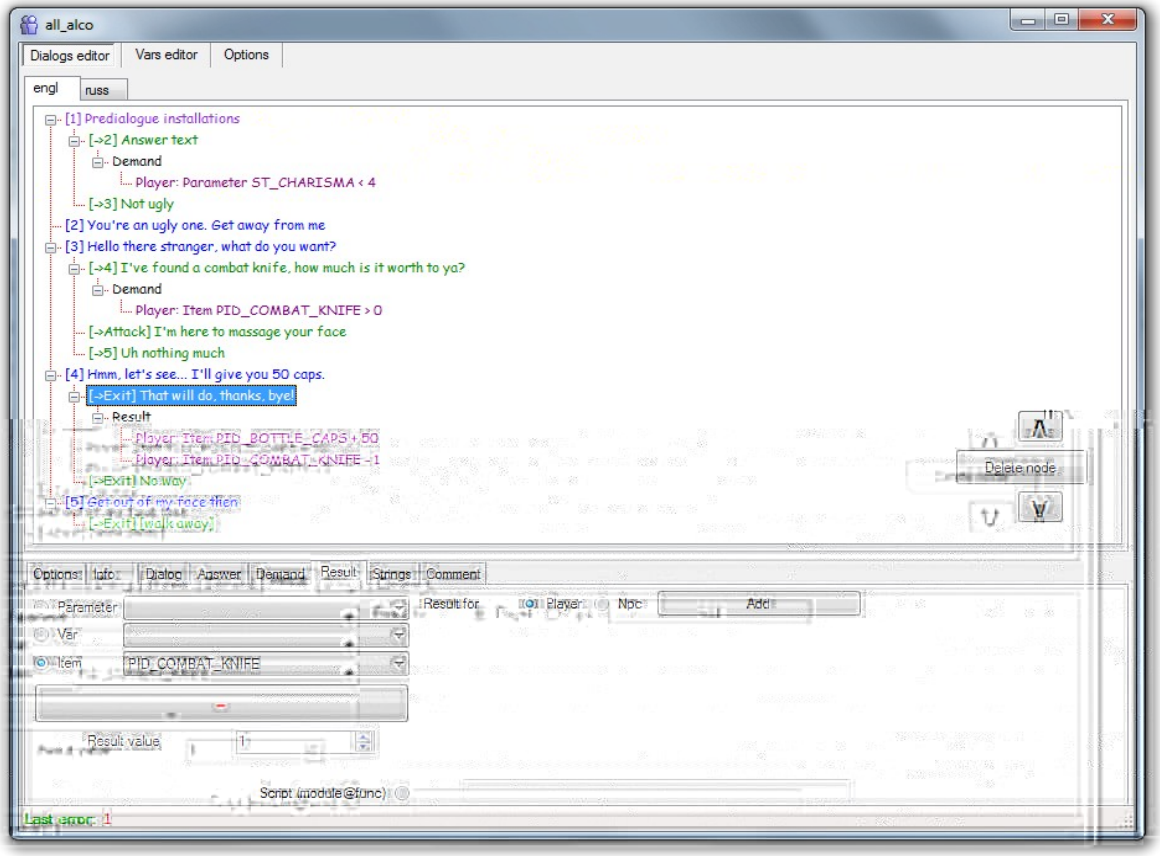
It's always a good idea to give the player a way to decline an offer. Giving only one answer option to the player could be considered bad design, depending on what kind of dialog you're making.

Another thing that should be said is that you can have multiple answers and demand on the same node. In the case of demands, **all demands** will be required for the answer to be available.

Likewise, **all results** will run when choosing the answer node that they are attached to.

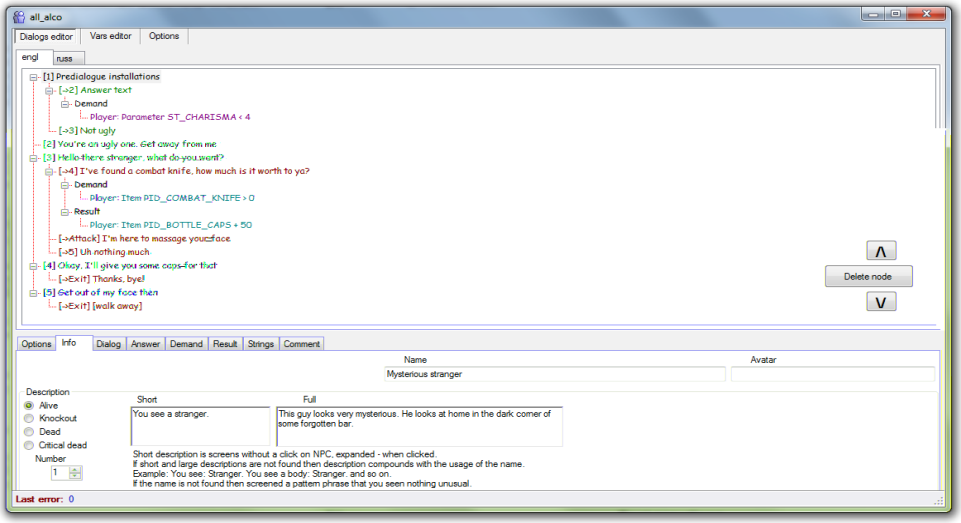
We'll round up with creating two more answers, one saying “I'm here to massage your face” with an *Attack* attached to it and the other, “Uh nothing much”, pointing to the node “Get out of my face then” which we have to create an additional answer to with the text “[walk away]” which we give the option *Close dialog*.

The dialog tree should now look something like this:



Now we can save the dialog as something else than all_alco and we are done. This concludes the step-by-step guide. Of course there's more things to talk about.

Descriptions



When we create a new dialog, we always want to give the NPCs carrying the dialog a specific name a description. This section is pretty straightforward, here you enter the short/full descriptions and name that will appear when player hoovers mouse over him/her. To change description, click *the info tab*.
It is possible to have multiple descriptions, tick the number arrows to decide on more descriptions (fully optional).

Game variables

There are three types of variables, there's Global, Local and Unicum variables. Global variables have only one instance. Local variables are per critter (player or NPC). Which means that setting a local variable in the dialog, sets it **only** for the player or the NPC (depending on which one you've selected in result/demand), while a global will be the same for all players. Local variables or LVars as they are referred to, are by far the most common variables.

The Unicum variables are used when a variable is to be linked in a player-NPC fashion. One value is reserved for each pair (a combination of one player and one NPC). Let's say a player approaches a trader and initiates a conversation. If we were to use a Unicum value here, the value set in the dialog will be valid **only** for this unique pair. If the player approaches another trader, the value set will not be the same. Likewise, if another player approaches the first trader after the first player has talked with him and some Unicum variable has been set in the dialog, it will still not be stored in the same place.

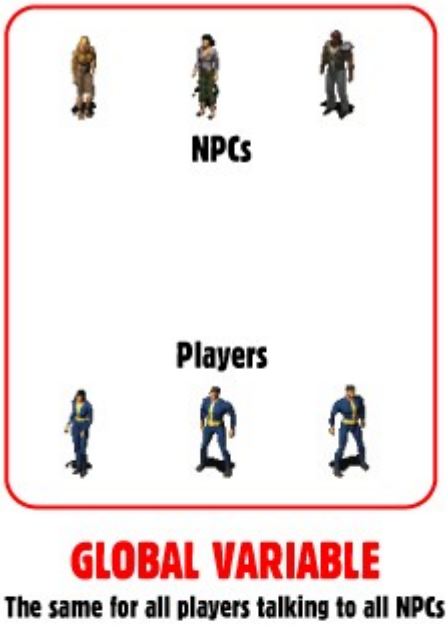
In summary, Unicum variables are set for a pair of player and NPC he talks to. Rather than belonging to a player or to an NPC like local variables, it belongs to an unique pair player-NPC (or player-player, NPC-NPC, but this never happens in dialogs, as you don't initiate dialogs with other players, nor does NPCs with each other).

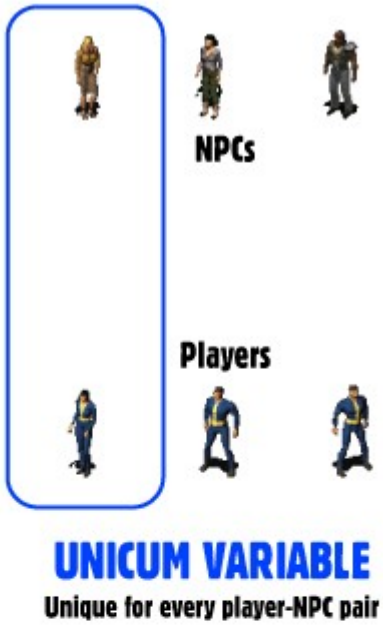
This is very useful when working with relationship based variables, like **name_mem_npc_player** which stores if the NPC has met the player before, or the **name_mem_player_npc** which inversely, keeps track if the player has met or conversed with the NPC before. The difference might be subtle, but it's recommended to use **name_mem_npc_player** over the other variant, as the deletion of a critter will automatically clear all its values. Also it's better for consistency.

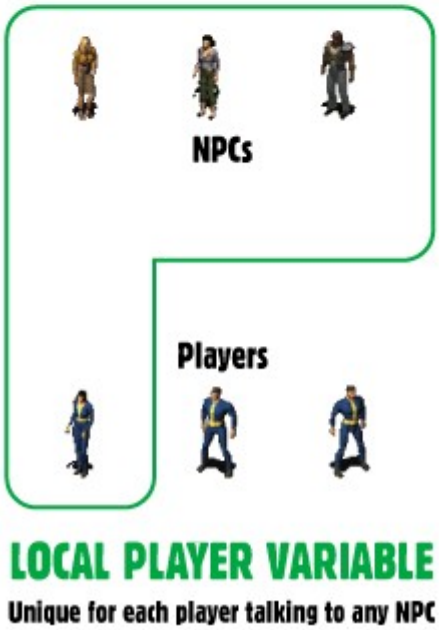
If you want to have different greetings or only have some nodes available the first time a

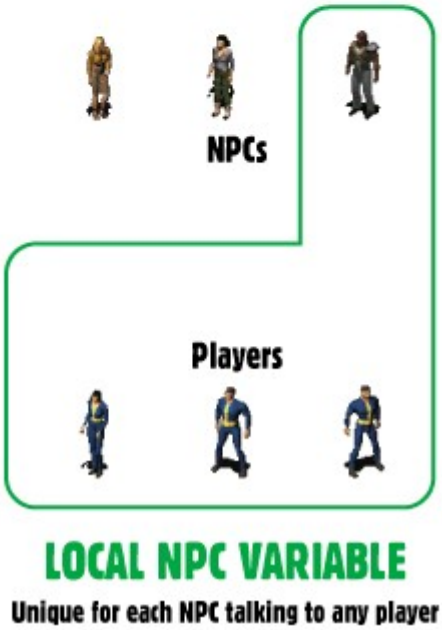
player is talking to an NPC you can use the **name_mem_npc_player** variable, simply check if it's 0 (with a demand) and if so, the player is unknown. In a node where you want to make the player known to the NPC, you set **name_mem_npc_player** to 1 with a result node attached to the answer node.

It might all seem very confusing, which it certainly was for me, the illustration below might make things a bit clearer.

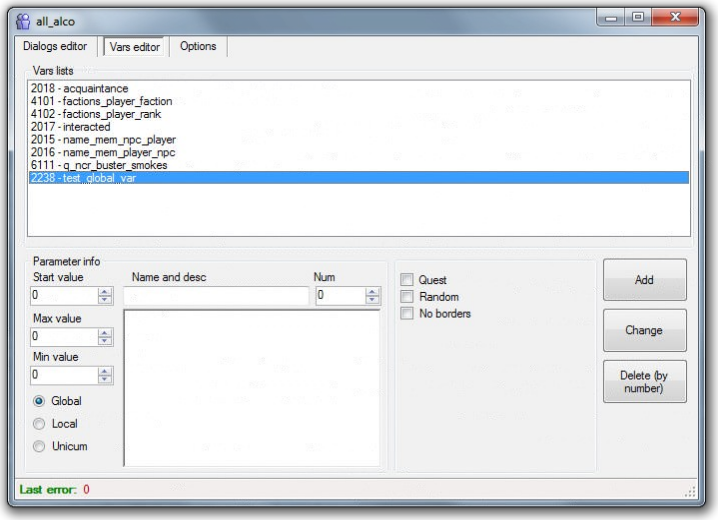








You can manage variables in the *vars editor* tab.



To add a new variable, enter a name and an id not used, choose among the options presented and *click add*.

Double-click on a variable to edit it, then *click change* to apply changes.

Start value is the variables default value. Max value is the maximum amount the value can reach, if it's 2 for example and 3 is set, it will be decreased down to 2 again.

- Min value works in the same way, but in the opposite direction.
- If your variables is used in a quest, check the *Quest checkbox*.
- *No borders* means that there's no min/max value used.
- The *Random checkbox* will assign a random start value.

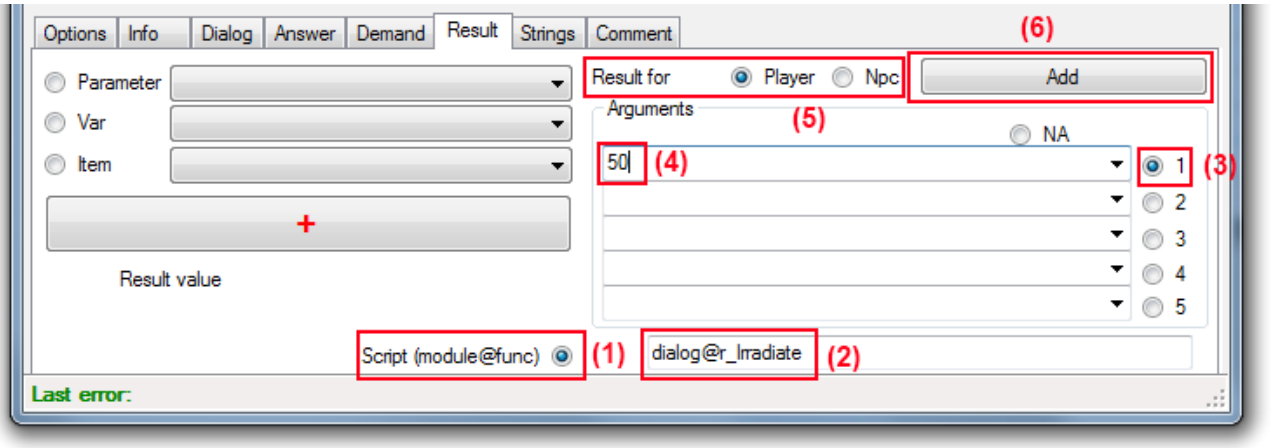
Using script functions

When creating a more advanced dialog, for a quest, we sometimes need to use functionality that doesn't inherently exist in the dialog editor. This is when we have to call actual game code. FOnline: 2238 has a number of functions tailored specifically for usage through dialogs.

These can be used both in demands and results. See the HTML page named dialogfuncs.html that should be included with this bundle.

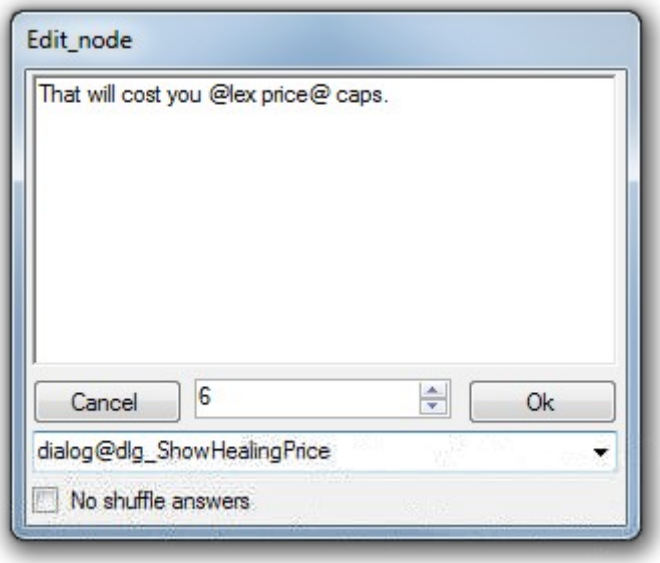
To use a dialog function we click *Script (module@func)* in either the demand or result tab. Now we must see if the function requires any arguments. Let's take a look at the function named d_CheckNight, The prefix tells us that it's a demand (results of course has r_ as prefix) it will demand that it's night for the node to be available. In the documentation it says simple “bool d_CheckNight ()”, we enter dialog@d_CheckNight into the *Script (module@func) field*. There is no arguments needed here which can be seen by the empty brackets. In the arguments field we choose *N/A (Not applicable)* and *click Add*.

Another function of interest is r_Irradiate (int val). This function requires one argument, namely the amount of radiation the player should receive (can be negative). As we have to supply an argument (dialog will fail otherwise), we press the radio button “1”. In the field *we enter our radiation amount*, then we *click add*.



Dlg functions are attached to dialog nodes used for processing “say text” and so called lexems which are values supplied into the dialog during runtime. To add a dlg function to a dialog node (not an answer node!), right-click the node and replace “None” with dialog@<name of the dlg function>.

The dlg function dlg_ShowHealingPrice (string @text) will replace a lexem (text symbol) in the node with the actual cost to heal for the player. For this to work it requires that you've include the lexem “price” somewhere in the node. For example: “This will cost you @lex price@ caps.” will replace @lex price@ with the price value by calling the script and requesting the price.



Lexems

There are a few hardcoded lexems that are driven by the engine. If you want some specific text to be written depending on sex, you can use the @sex@ lexem. An example of using this lexem would be “hello @sex@|brother||sister|”. The first is of course the text that will be shown for males and the second the alternative text that will be shown for female players.

Another useful lexem that are used a bit more rarely is @pname@ it replaces the lexem with the name of the player initiated in the conversation.

Say

If you want use the say function, you unfortunately need to use scripts for parsing the input in a more meaningful way. The text entered is passed to the scripting the function where the code can decided what to do with the input.

Comments

In the comments tab you can write some descriptive comment about the dialog.

Best practices

It's always useful to follow the best practices, which are called just that for a reason. If variable naming and dialog naming is consistent it's much easier to understand for what reason it's there and what purpose it serves, this is why we absolutely recommend you to follow these guidelines when creating dialogs.

Dialog naming

Although not entirely as consistent as we want it to be, there are some rules for how to name your dialog files.

- If it's a dialog used in several locations (for example in random encounters) it should be prefixed **all**. An example would be all_canibal.fodlg.
- If it's a dialog used in only one location, the **location prefix** should be used, and in some specific cases the sub location (very rarely used). An example of this would be ncr_citizen.fodlg. Now you're of course asking, “which are the location prefixes”, a list has been assembled below.
- In some special cases, a special prefix is used when there's a special functionality used across several locations, an example is economy_banker.fodlg.
- **q** is used as prefix in cases where it belongs to a quest character that is present in a spawned quest location, and not in an ordinary static location. An example is q_tanker_terminal.fodlg

List of dialog prefixes:

- bh – Broken Hills
- bos_lh - Lost Hills
- cath - Cathedral
- den – Den
- geck - Gecko
- gun - Gun Runners (In Boneyard)
- hub – Hub
- junktown – Junktown
- klam – Klamath
- la_ady – Adytum
- la_blade – LA (Blades old HQ)
- la_lib – LA library
- nav - Navarro
- ncr – New California Republic

- necropolis – Necropolis
- nr – New Reno
- raiders – Raiders HQ
- redd – Redding
- san – San Franscisco
- vaul – Vault City

Quest rewards

When you're giving out quest rewards, you should always do it in the same node as where you set the quest variable to its final value. The reason for this is that if you set it later, the player might click the reward node to get the rewards and then simply force exit the dialog by pressing escape or '0' before having clicked the “good-bye” answer that's actually marking the quest as finished. This can lead to dangerous infinite XP/item exploits.

Last words

A lot of things might still seem like a mystery, even after this tutorial. I highly suggest that you check all the sample dialogs and see how they look. It's often said that learning by example is the fastest way.

If you're looking for some additional guidelines when it comes to structuring your dialogs, I suggest you check out the dialog tutorial written by **Pjnt** for *Fan Made Fallout* (a Fallout 2 mod no longer in development), it contains some general tips that are relevant to FOnline as well.

J.E Sawyer also has some quick tips to give in one of his blog posts.

I hope that you're now feeling a bit more comfortable in the role of a dialog writer.

Credits

Tutorial written by Ghosthack
DialogEditor created by Cvet
Suggestions & Additional Graphics by JovankaB

I would like to thank the members of Rotators and Cvet for all their efforts.

Thanks to the original Black Isle Studio team for making Fallout.